Lecture 2

Mark-Jan Nederhof

1 analysis

Recapitulation

Regular languages (FAs) allow linear-time processing

CF processing is more expensive (generally cubic-time)

Not possible in general to do full CF processing with finite automata

Undecidable when this can be done

But is there sufficient condition for CFG to generate regular language?

Can we approximate CFG by FA when the sufficient condition does not hold?

Analysis of CFG

We partition N into k maximal sets of *mutually recursive nonterminals*:

$\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$

$N_1 \cup \cdots \cup N_k = N$

$\forall i[N_i \neq \emptyset]$

$\forall i, j[i \neq j \Rightarrow N_i \cap N_j = \emptyset]$

For all $A, B \in N$:
$\exists i[A, B \in N_i] \quad \Leftrightarrow \quad \exists \alpha_1, \beta_1, \alpha_2, \beta_2[A \Rightarrow^* \alpha_1 B \beta_1 \wedge B \Rightarrow^* \alpha_2 A \beta_2]$

Note that if A is not recursive, then $\exists i[N_i = \{A\}]$

Cf. **strongly connected component** in directed graph

Self-embedding CFGs

For the respective $N_i \in \mathcal{N}$ define:

$$\begin{array}{rcl} \mathit{leftgen}(N_i) & = & \exists(A \rightarrow \alpha B \beta)[A, B \in N_i \wedge \alpha \neq \varepsilon] \\ \mathit{rightgen}(N_i) & = & \exists(A \rightarrow \alpha B \beta)[A, B \in N_i \wedge \beta \neq \varepsilon] \end{array}$$

$$\mathit{recur}(N_i) = \begin{cases} \mathit{left} & \text{if } \neg\mathit{leftgen}(N_i) \wedge \mathit{rightgen}(N_i) \\ \mathit{right} & \text{if } \mathit{leftgen}(N_i) \wedge \neg\mathit{rightgen}(N_i) \\ \mathit{self} & \text{if } \mathit{leftgen}(N_i) \wedge \mathit{rightgen}(N_i) \\ \mathit{none} & \text{if } \neg\mathit{leftgen}(N_i) \wedge \neg\mathit{rightgen}(N_i) \end{cases}$$

We say CFG is **self-embedding** (s.e.) if $A \Rightarrow^* \alpha A \beta$, for some A, $\alpha \neq \varepsilon$, $\beta \neq \varepsilon$

CFG is s.e. iff $\mathit{recur}(N_i) = \mathit{self}$ for at least one set N_i

When can we tell CFL is regular?

Sufficient condition for regularity

If CFG is not self-embedding, then it generates regular language

(Chomsky, Information and Control 2, pp. 393-395, 1959)

Alternative proof by construction of FA out of non-self-embedding CFG

Create initial state q_0 and final state q_f

Call $\mathit{make\_fa}(q_0, S, q_f)$

Construction of FA from non-s.e. CFG

A call $\mathit{make\_fa}(q, \varepsilon, q')$ leads to:

- addition of epsilon transition (q, ε, q')

A call $\mathit{make\_fa}(q, a, q')$ leads to:

- addition of transition (q, a, q')

A call $\mathit{make\_fa}(q, X\alpha, q')$, $X \in N \cup \Sigma$ and $\beta \neq \varepsilon$, leads to:

- creation of new state q''
- call $\mathit{make\_fa}(q, X, q'')$ and call $\mathit{make\_fa}(q'', \alpha, q')$

Yet to define is implementation of call $\mathit{make\_fa}(q, A, q')$

Which depends on value of $\mathit{recur}(N_i)$, for N_i with $A \in N_i$

Construction of FA from non-s.e. CFG (cont.)

With $recur(N_i) = left$, $A \in N_i$, a call $make\_fa(q, A, q')$ leads to:

- creation of new state q_B for each $B \in N_i$
- addition of epsilon transition (q_A, ε, q')
- call $make\_fa(q, X_1 \cdots X_m, q_C)$ for each $C \to X_1 \cdots X_m$ such that $C \in N_i$ and $X_1, \ldots, X_m \notin N_i$
- call $make\_fa(q_D, X_1 \cdots X_m, q_C)$ for each $C \to DX_1 \cdots X_m$ such that $C, D \in N_i$ and $X_1, \ldots, X_m \notin N_i$

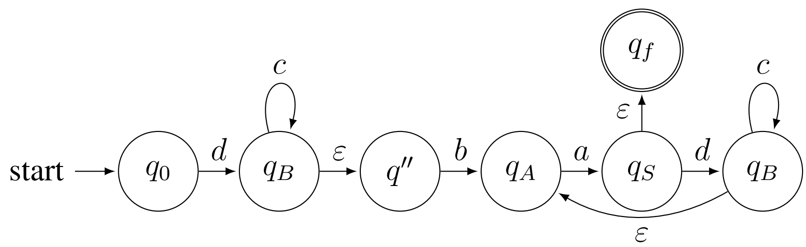
With $recur(N_i) = right$, the situation is symmetric

The case $recur(N_i) = none$ can be treated arbitrarily as $recur(N_i) = left$ or $recur(N_i) = right$

Example

$$
\begin{array}{lcl}
S & \to & A\ a \\
A & \to & S\ B \\
A & \to & B\ b \\
B & \to & B\ c \\
B & \to & d
\end{array}
$$

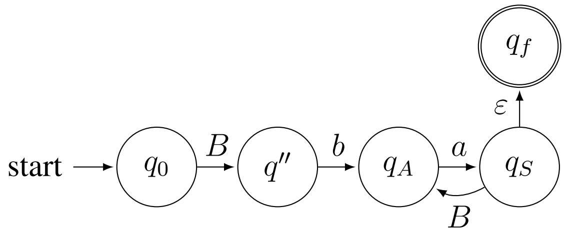
$$
\begin{array}{ll}
\mathcal{N} = \{N_1, N_2\} & \\
N_1 = \{S, A\}, & recur(N_1) = left \\
N_2 = \{B\}, & recur(N_2) = left
\end{array}
$$



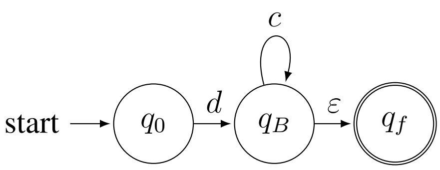
Compact representation

Or build compact representation, where subautomaton for each nonterminal is constructed only once

E.g. subautomaton for S:



Subautomaton for B:



Advantage of compact representation

Each subautomaton can be determinised and minimised individually

Then subautomata are substituted in each other

(**regular substitution**)

And determinised and minimised once more

This keeps intermediate automata small

2 approximation

Approximation

What if $recur(N_i) = self$ for at least one $N_i \in \mathcal{N}$?

Then language may not be regular

One can **approximate** such N_i by FA

For other $N_j \in \mathcal{N}$ with $recur(N_j) \in \{left, right, none\}$ construct (exact) subautomata as before

Combine exact subautomata and approximate subautomata, much as before

Superset approximation: FA accepts superset of CFL

Subset approximation: FA accepts subset of CFL

Different methods of approximation

Superset:

- RTNs
- Pushdown automata (PDAs)
- n-grams

Subset:

- Restricting recursion
- PDAs

and several more that are not discussed here

Superset approximation by RTNs

Recursive transition network (RTN) is generalisation of CFG

Instead of rules with LHS A, there is one finite automaton for each A

Transitions therein labelled by symbols in $\Sigma \cup N$

Following transition labelled with nonterminal is like calling subroutine

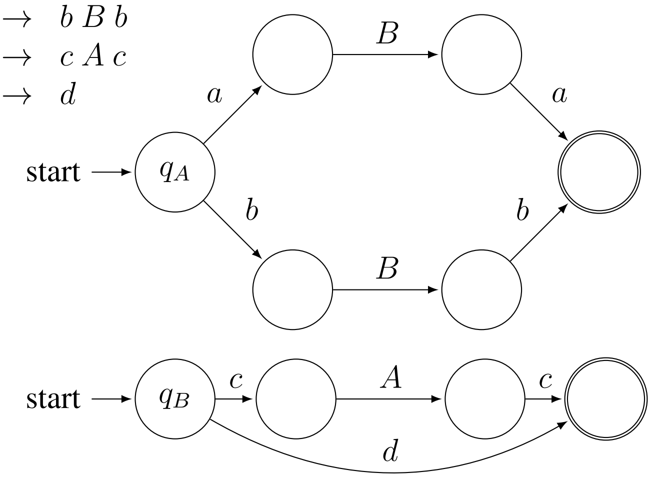
After completion of subroutine, return to state after call

Hence ’recursive’ in RTN

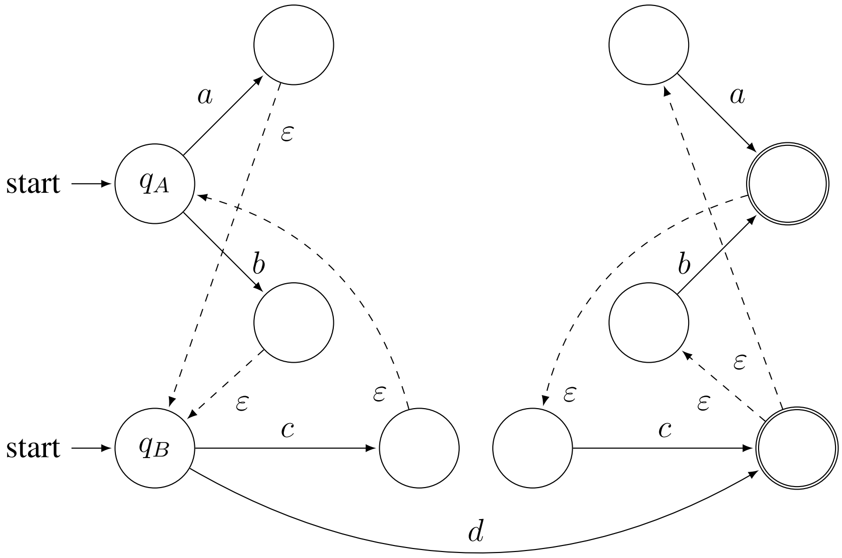
Approximation by forgetting which call we came from

Example RTN

$$\begin{array}{lcl} A & \rightarrow & a\ B\ a \\ A & \rightarrow & b\ B\ b \\ B & \rightarrow & c\ A\ c \\ B & \rightarrow & d \end{array}$$



Example: its approximation



Refinement

We have thrown away all history upon call of subroutine

Can we do better?

Given a bound k (small number)

Remember history of up to k preceding calls of subroutines

So that we can return to the correct state after a call

Finite history can be encoded in FA states

Subset approximation by blocking self-embedding

Intuition:

Block s.e. derivations $A \Rightarrow^* \alpha A \beta$, $\alpha \neq \varepsilon$, $\beta \neq \varepsilon$

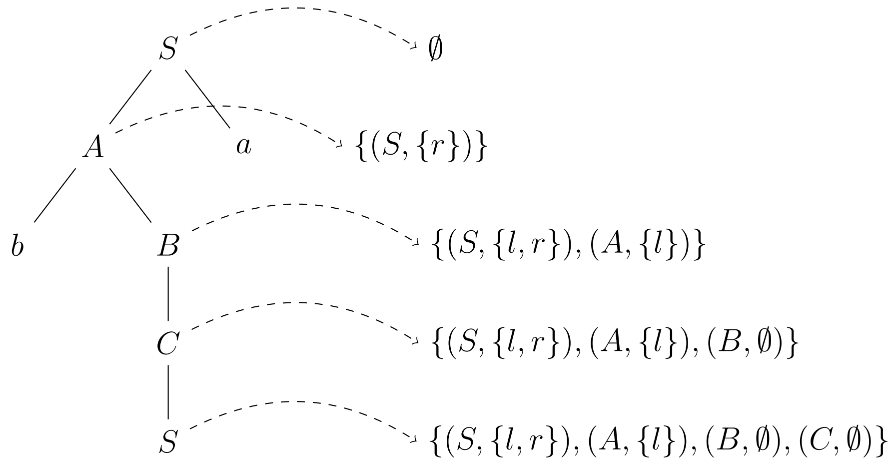
Method:

For each A, keep track of material generated to the left (l) and right (r) of previous occurrences of A *in same spine*

This is bounded information and can be encoded in enriched nonterminal names

Disallow further derivation once we have $\{l, r\}$ for a recurring nonterminal

Example



Here block due to s.e.

More precisely

Enhanced nonterminals A^F, where F is set of pairs (B, Q) with $B \in N$, $Q \subseteq \{l, r\}$

If original grammar contains $A \to w_0 A_1 w_1 \cdots A_m w_m$

Transformed grammar will have $A^F \to w_0 A_1^{F_1} w_1 \cdots A_m^{F_m} w_m$ where for $1 \leq j \leq m$:

- $F_j = \{(B, Q \cup Q_l^j \cup Q_r^j) \mid (B, Q) \in F \cup F'\}$;
- $F' = \emptyset$ if $\exists Q[(A, Q) \in F]$ and $F' = \{(A, \emptyset)\}$ otherwise;
- $Q_l^j = \emptyset$ if $w_0 A_1 w_1 \cdots A_{j-1} w_{j-1} = \varepsilon$ and $Q_l^j = \{l\}$ otherwise;
- $Q_r^j = \emptyset$ if $w_j A_{j+1} w_{j+1} \cdots A_m w_m = \varepsilon$ and $Q_r^j = \{r\}$ otherwise.

And F should not contain $(A, \{l, r\})$

Variants

Many variants of this subset approximation are possible

E.g.:

- Allow up to k self-embeddings, small fixed k
- Simplify enhanced nonterminals (not distinguish between different nonterminals seen before)
- Combinations of these two variants

Superset approximation with PDAs

Pushdown automaton (PDA) can be seen as FA, but enhanced with pushdown store (stack)

Superset approximation results by:

- Ignoring the stack, preserving only the internal states
- Also remembering the top-most k stack elements (small fixed k)
- Other 'congruence relations' on stacks, partitioning the infinite set of possible stacks into finite number of classes

Several such methods were proposed based on **LR parsing**

Subset approximation with PDAs

Method:

Construct PDA from CFG by some parsing strategy

Place bound on height of stack

Turn each possible stack into FA state

Often used for this is (modified) **left-corner parsing**

Is partly top-down and partly bottom-up strategy

If grammar is non-s.e. then the stack height is naturally bounded

Superset approximation based on n-grams

Given CFG, the set of all substrings of length n of strings in the language can be effectively computed

Each such substring becomes state in FA

We have transitions (aw, b, wb), where aw and wb are possible substrings of length $n - 1$

(Special cases at beginning and end of string)

3 natural language

Self-embedding in natural language

The luggage arrived

The luggage that the passengers checked arrived

The luggage that the passengers that the storm delayed checked arrived

$\vdots$

This construction can be modelled by the schema $a^n b^n$, with $n = 1, 2, \ldots$, where 'a' stands for '**that the** *Noun*' and 'b' stands for any transitive verb

Can be described by self-embedding CFG

Competence versus performance

Sentences with several levels of self-embedding are considered grammatical but hard to understand

Here lies motivation to place bound on levels of self-embedding (cf. some of our subset approximations)

Competence grammar: idealised language processing

Includes any level of self-embedding

Performance grammar: restricted by mental capacity

Places (soft) bound on levels of self-embedding

Consider also cross-serial dependencies

Experiments

Experiments with several subset and superset approximations were reported by:

Nederhof, *Computational linguistics* 26(1), pp. 17-44, 2000

Extraction of NLP grammars from parts of a treebank

Parts of different sizes, resulting in grammars of different sizes

Measured:

- sizes of resulting FAs, for different methods
- how many ungrammatical sentences are allowed by superset approximations
- how many grammatical sentences are rejected by subset approximations

Experiments: some outcomes

Subset approximations tend to lead to huge FAs, growing very fast for increasing size of CFG

Of superset approximations, the 2-gram and the RTN methods give automata of relatively small sizes

RTN method tends to be more precise than 2-gram method (smaller superset)